CARLETON UNIVERSITY

SCHOOL OF COMPUTER SCIENCE

COMP4905 – HONOURS PROJECT

Formation Flight of UAVs Swarms in an Obstacle–Filled Three–Dimensional Environment

Author
Yannick ABOUEM

Supervisor
Prof. Mark LANTHIER

July 31, 2024

Abstract

Abstract goes here!

Acknowledgements

This project uses Webots (http://www.cyberbotics.com), an open-source mobile robot simulation software developed by Cyberbotics Ltd.

Contents

List of Figures

List of Algorithms

1 Introduction

Cooperation among different individuals in a group is often necessary to solve complex problems that are otherwise harder or impossible to complete for a singular individual. This concept is true in the field of robotics, where robotic swarms systems, which are systems composed of numerous small and limited robots (Oh et al. 2017), are a crucial area of study. Due to the early stage of development of this field, there are not many currently existing applications, however there exists many research projects to test the capabilities of swarm algorithms (Schranz et al. 2020). Despite the lack of representation in the industry sector, multi-robot systems have become of interest recently due to their abilities to resist failures and damages, adaptability to new environments and low costs (Oh et al. 2017).

A specific problem in swarm robotics, is the pattern formation problem, which consists of "getting a group of robots to form and stay in a specific formation, like a wedge or a chain, and maintaining that formation" (Sri and Suvarchala 2022). The robots in the swarm need to coordinate to maintain a specific shape in order to achieve the desired goal. In this process, different methods of controlling the swarm can be used, for example the election of a leader or the imposition of a certain set of behaviors for each member. There is also a distinction between centralized and decentralized pattern formation, where centralized pattern formation is performed when there exists a centralized unit that coordinates the individual robots (Oh et al. 2017).

In this project we will explore a decentralized and distributed solution to the pattern formation problem. A distributed approach was chosen because of their robustness to failure, due to their built-in redundancy since each robot in the swarm receives the same role (Oh et al. 2017). This is advantageous compared to a centralized, non-distributed approach as it mitigates the possibility of a single point of failure.

1.1 Swarms of Unmanned Aerial Vehicles

A swarm of Unmanned Aerial Vehicles is a swarm of robots where each robot is capable of flight using a set of rotors. These swarms have multiple uses. They are commonly used in military applications, but they are also used for civilian applications, such as aerial surveys, disaster management, environment mapping, search and rescue and for leisure (Tahir et al. 2019). These are applications that would benefit from pattern formation, as we can direct the swarm to take an optimal shape for its use case. For example, we can choose a shape to best cover an area while performing an aerial survey

or environment mapping, or adopt a chain-like formation to establish communication between two points (Schranz et al. 2020) which can be deployed at a site of a natural disaster to aid in search and rescue efforts.

1.2 Objectives

The objective of this project is to provide a distributed solution to the pattern formation problem, meaning build a system of multiple robots capable of positioning themselves in a specific shape and maintain this shape while navigating an environment dotted by obstacles. This will be achieved using obstacle detection and avoidance techniques, as well as distributed system principles, such as leader election, and computer network principles. For this project we chose to use unmanned aerial vehicles to demonstrate the efficacy of the solution. Similar work being done includes (Chen et al. 2021), where a hierarchical approach is applied to formation control of fixed wing UAVs, by breaking up the swarm into several smaller groups headed by a leader.

2 Methodology

In this chapter we will explore the method used in and considered for this project. Since this project contains multiple sub-problems we decided to divide this chapter into four sections.

2.1 Automated flight

The first consideration for this project is about the autonomous flight of the UAVs, meaning that there should not be any input from the user regarding the flight control. This problem can then be broken down into two components: the control algorithm that controls the ability of the UAV to fly and the navigation algorithm that controls the direction of flight.

2.1.1 Control Algorithm

Flight controllers can be subdivided into two categories: linear controllers and non-linear controllers. Linear controllers use linear methods to control flight, which includes conventional controllers such as PID, H_∞ controller, gain scheduling and Linear Quadratic Regulator. While non-linear controller are derived from the original dynamic model of the UAVs and arose to overcome shortcomings of linear controllers. They are generally harder to implement. Some examples are Feedback linearization, backstepping, sliding mode control, adaptive control, and model predictive control. (Nguyen et al. 2020)

For this project we settled on linear controller due to their simplicity to implement, in particular we considered the PID controller and the LQR controller.

PID Control

The Proportional Integral Derivative controller is the most common controller used in industrial settings due to its usefulness and ease of implementation. (Lopez-Sanchez and Moreno-Valenzuela 2023) The controller is composed by a proportional element, an integral element and a derivative element, that represent the present, past and future error value(Araki 2009). It is usually designed in the structure represented in Fig. 2.1.

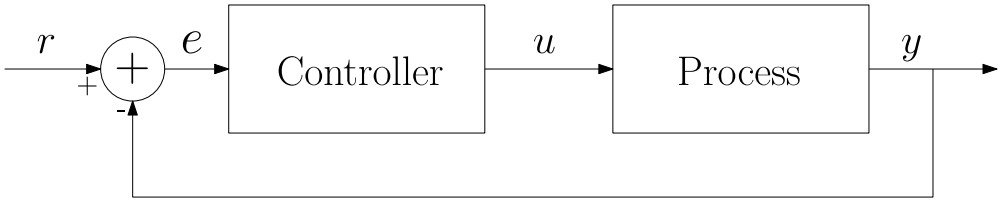


Figure 2.1: Diagram of a conventional PID controller

The controller first computes the error at a specific time t ($e(t)$) using Eq. 2.1.

$$e(t) = r(t) - y(t) \tag{2.1}$$

Where $r(t)$ is the set value and $y(t)$ is the real value of the process (Araki 2009). Form the computed error the controller then finds a control factor $u(t)$ using a proportional coefficient (K_p), an integral coefficient (K_I) and a derivative coefficient (K_D) using Eq. 2.2 (Nguyen et al. 2020).

$$u(t) = K_p e(t) + K_I \int_0^t e(t)\mathrm{d}t + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{2.2}$$

In this project we decided to implement a PID controller due to the ease of implementation and the higher number of resources available at our disposal.

LQR Control

The other option considered, is the Linear-Quadratic Regulator controller (LQR controller), which is the optimal linear controller to operate a dynamic system at minimum cost (Nguyen et al. 2020).

$$J_{LQR} = \int_0^\infty y_i^T(t) Q y_i(t) + u_i^T(t) R u_i(t)\mathrm{d}t \tag{2.3}$$

Eq. 2.3 computes the minimum quadratic cost (J_{LQR}) from the control input u_i and control output y_i, using two weight matrices (Q and R) (Nguyen et al. 2020).

2.1.2 Navigation Algorithm

For the navigation algorithm we used a simple algorithm where given a waypoint, we compute the derivative component of the PID algorithm. This can be achieved with the following algorithm.

Algorithm[2] 1 Simple algorithm for UAV navigation using PID

Input: u, v, Y
Output: Y_d, P_d

$\alpha \leftarrow ((\text{atan2}(u_y - v_y, u_x - v_x) - Y) + 2\pi) \text{mod}(2\pi)$
if $\alpha > \pi$ **then**
    $\alpha \leftarrow \alpha - 2\pi$
end if
$Y_d \leftarrow K_{Yd_{max}} \frac{\alpha}{2\pi}$
$P_d \leftarrow \log(|\alpha|)$
if $P_d < K_{Pd_{max}}$ **then**
    $P_d \leftarrow K_{Pd_{max}}$
else if $P_d > 0.1$ **then**
    $P_d \leftarrow 0.1$
end if
return Y_d, P_d

Where u and v are vectors representing the UAV position and the waypoint position, Y is the current yaw angle of the UAV, Y_d and P_d are the yaw angle disturbance and the pitch angle disturbance respectively, and $K_{Yd_{max}}$ and $K_{Pd_{max}}$ are the maximum allowed disturbance of the yaw and pitch respectively.

2.2 Obstacle Detection

There exist a wide array of sensors used in obstacle detections, some of these include passive sensors like cameras and infrared sensors, and active sensors like lidar, radar and ladar. Passive sensors absorb energy, like light or infrared waves, to create a picture of the environment while active sensors emit their own energy then collect the reflection emitted by the environment. Active sensors are better suited for our application, as they do not depend on atmospheric conditions or lighting condition to operate. However, active sensors are prone to interference from other sources of electromagnetic waves. (Discant et al. 2007) For this project we decided to equip the UAVs with radar to detect obstacles.

Webots offers a simulation of a radar sensor which detects targets in the environment and provides their distance from the sensor, the power received back from the target, the speed relative to the sensor, and the horizontal angle (the azimuth) between the target and the sensor (Webots n.d.). However, simply knowing this information about the obstacle is not sufficient as we need to find its position in order to avoid it.

[2]Adapted from (Cyberbotics 2023) source code

2.2.1 Target Coordinates Extraction

To compute the position of a target we simply need its distance from the sensor, d, and the azimuth, α from the values returned from Webots API. Assuming that the UAV position is located at the origin, the distance between the UAV and the target can be considered a vector with magnitude d and the azimuth is the angle between the distance vector and the direction vector of the UAV. We call these vectors v and u respectively. Thus, what we are trying to find is $v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$.

Given that we have two vectors and the angle between them, we can then find their dot product, where $\| v \| = d$, using Eq. 2.4.

$$u.v = \| u \| \| v \| \cos\alpha = u_x v_x + u_y v_y \tag{2.4}$$

Assuming u is a unit vector, since u is the direction of the UAV then the angle β between the vector u and the x-axis can be calculated from the heading of the UAV. Thus, we can compute the components of u using Eq. 2.5 and Eq. 2.6.

$$u_x = \cos\beta \tag{2.5}$$
$$u_y = \sin\beta \tag{2.6}$$

Therefore, we can rewrite Eq. 2.4 combined with the values of u_x and u_y resulting in Eq. 2.7.

$$\| v \| \cos\alpha = v_x \cos\beta + v_y \sin\beta \tag{2.7}$$

Similarly to u, we can also find the components of v using the angle between v and the x-axis, γ which is equal to $\beta - \alpha$. Using this angle we can write an equation to find the components of v.

$$\tan\gamma = \frac{v_y}{v_x} \tag{2.8}$$

Eq. 2.8 can then be written in terms of v_y resulting in Eq. 2.9.

$$v_y = v_x \tan\gamma \tag{2.9}$$

Finally, we can combine Eq. 2.7 with Eq. 2.9 to create one equation to find the x component of v.

$$v_x = \frac{d\cos\alpha}{\cos\beta + \tan\gamma \sin\beta} \tag{2.10}$$

Solving Eq. 2.10 and Eq. 2.9 it will produce the coordinates of the target relative to the UAV. From there it is simply necessary to translate the vector v by the UAV position.

2.3 Obstacle Avoidance

Once obstacles are detected, the UAVs should be capable of avoiding them and navigate between them until a goal is reached. As explained above, the navigation algorithm used in this project computes the trajectory to take to reach a specific waypoint, so we can define the path the UAVs have to take as a list of waypoints. If we restrict the waypoints as being intersections of cells in a grid then we can abstract this grid as being a graph. Thus, what we wish to do, in this case, is to find a path from a starting node to a predefined end node, or search the graph for a specific node.

There exist many algorithms capable of solving this problem, with the most popular being A*.

2.3.1 A*

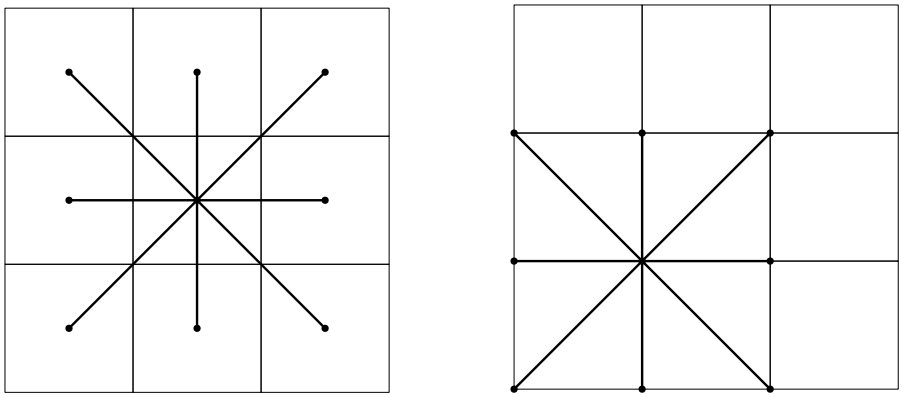
The most obvious and popular solution is the A* algorithm, which improves on Dijikstra's algorithm by using heuristics to decide which node should be considered next (*A* Search Algorithm* n.d.). However, since we are working in an unknown environment where the UAVs gain more knowledge of the obstacles locations as they progress, the path needs to be re-planned each time a new obstacle is found. This can become computationally expansive as we need to execute A* multiple times.

2.3.2 Path Planning in Partially-Known Environments with D*

In conditions where the UAVs do not have complete information about their surroundings it is advantageous to use an algorithm that assumes the possibility of potential changes in costs of the nodes. One of such algorithms is D*.

D* is capable of generating optimal paths for sensor equipped robots with a map of the environment that can be complete, partial or empty. The algorithm formulates the path planning process as a set of states, that represent potential robot locations, connected by directional arcs each with an associated cost. The algorithm maintains a list of states which cost has changed over time, and each time a cost is changed the affected states are added onto the list. Instead of re-planning the whole path, D* adjusts the path from the states where a change in cost happened and subsequent states. Thus, it operates faster in real-life scenarios than an algorithm that re-plans the whole trajectory each time a new obstacle is found. (Stenz 1994)

Despite its advantage over other algorithms, it is still possible to improve over D*. For example, D* Lite claims to be as efficient if not better than D* but with a simpler implementation (Koenig and Likhachev 2005).



(a) Representation used by standard algorithms (b) Representation used by Field D*

Figure 2.2: Representation of nodes positioning in standard path planning algorithms and field D*

2.3.3 Field D*: Improving D* Lite Using Interpolation

For this project we decided to implement Field D*, due to its promises of minimizing unnecessary turns and generating less expensive paths than D* and D* Lite. Field D* achieves this by using interpolation in the computation of the cost of the path. (Ferguson and Stentz 2006)

Most path planning algorithms that use a grid as underlying graph compute the cost of a node s, $g(s)$, using Eq. 2.11.

$$g(s) = \min_{s' \in nbrs(s)} [c(s, s') + g(s')] \tag{2.11}$$

Where $nbrs(s)$ is the set of neighbors of node s, $c(s, s')$ is the cost of traversing the edge between s and s', and $g(s')$ is the path cost of s'. This assumes that the only possible path is a straight line between the nodes, resulting in limitations as described by (Ferguson and Stentz 2006). In order to overcome these limitations, Field D* finds an approximation of the cost of the path to any cell boundary node s_b using linear interpolation. To do this, the algorithm assumes that the nodes of the graph are not located at the center of the cells as shown in Fig. 2.2a but rather at the vertices as shown in Fig. 2.2b (Ferguson and Stentz 2006). It then follows Algorithm 2 to compute the path cost, v_s, of a node s given two of its neighbors s_a and s_b.

Field D* is not a faster algorithm than D* Lite, with a planning time 1.7 times slower than its non-interpolation-based counterpart, but it offers a more economical path, 96% as costly as D* Lite on average (Ferguson and Stentz 2006). Its path generation efficiency makes this algorithm competitive in some settings where minimizing the path cost is the main goal, for example in situation where the cost factor is the amount of fuel required to traverse an area.

Algorithm 2 Field D* path cost computation procedure by (Ferguson and Stentz 2006)

Input: s, s_a, s_b
Output: v_s

if s_a is diagonal neighbor of s **then**
  $s_1 \leftarrow s_b$
  $s_2 \leftarrow s_a$
else
  $s_1 \leftarrow s_a$
  $s_2 \leftarrow s_b$
end if
c is traversal cost of cell with corners s, s_1, s_2
b is traversal cost of cell with corners s, s_1 but not s_2.
if $\min(c, b) = \infty$ **then**
  $v_s \leftarrow \infty$
else if $g(s_1) \leq g(s_2)$ **then**
  $v_s \leftarrow \min(c, b) + g(s_1)$
else
  $f \leftarrow g(s_1) - g(s_2)$
  if $f \leq b$ **then**
    if $c \leq f$ **then**
      $v_s \leftarrow c\sqrt{2} + g(s_2)$
    else
      $y \leftarrow \min\left(\frac{f}{\sqrt{c^s - f^s}}, 1\right)$
      $v_s \leftarrow c\sqrt{1 + y^2} + f(1 - y) + g(s_2)$
    end if
  else
    if $c \leq b$ **then**
      $v_s \leftarrow c\sqrt{2} + g(s_2)$
    else
      $x \leftarrow 1 - \min\left(\frac{b}{\sqrt{c^2 - b^2}}, 1\right)$
      $v_s \leftarrow c\sqrt{1 + (1 - x)^2} + bx + g(s_2)$
    end if
  end if
end if
return v_s

2.4 Pattern Formation

Bibliography

A Search Algorithm* (n.d.). Webpage. Ada Computer Science. URL: https://adacomputerscience.org/concepts/path_a_star?examBoard=ada&stage=all (visited on 07/28/2024).

Araki, M. (2009). "PID Control". In: *Control Systems, Robotics, and Automation*. Ed. by Heinz Unbehauen. 2nd ed. UNESCO-EOLSS, pp. 58–65.

Chen, Hao et al. (2021). "Formation flight of fixed-wing UAV swarms: A group-based hierarchical approach". In: *Chinese Journal of Aeronautics* 34.

Cyberbotics (2023). *Maveric 2 Pro Patrol Controller*. source code. URL: https://raw.githubusercontent.com/cyberbotics/webots/master/projects/robots/dji/mavic/controllers/mavic2pro_patrol/mavic2pro_patrol.py.

Discant, Anca et al. (2007). "Sensors for Obstacle Detection - A Survey". In: *2007 30th International Spring Seminar on Electronics Technology (ISSE)*, pp. 100–105. DOI: 10.1109/ISSE.2007.4432828.

Ferguson, Dave and Anthony Stentz (2006). "Using Interpolation to Improve Path Planning: The Field D* Algorithm". In: *Journal of Field Robotics* 23.2, pp. 79–101. DOI: 10.1002/rob.20109.

Koenig, Sven and Maxim Likhachev (June 2005). "Fast Replanning for Navigation in Unknown Terrain". In: *IEEE Transaction On Robotics* 21.3, pp. 354–363.

Lopez-Sanchez, Ivan and Javier Moreno-Valenzuela (2023). "PID control of quadrotor UAVs: A survey". In: *Annual Reviews in Control* 56.

Nguyen, Hoa et al. (May 2020). "Control Algorithms for UAVs: A Comprehensive Survey". In: *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems* 7, p. 164586. DOI: 10.4108/eai.18-5-2020.164586.

Oh, Hyondong et al. (2017). "Bio-inspired self-organising multi-robot pattern formation: A review". In: *Robotics And Autonomous Systems* 91.

Schranz, Melanie et al. (2020). "Swarm Robotic Behaviors and Current Applications". In: *Frontiers in Robotics and AI*. doi: 10.3389/frobt.2020.00036. PMID: 33501204; PMCID: PMC7805972.

Sri, S.M. Lakshmi and C.V. Kavya Suvarchala (2022). "Multi-robot systems: a review of pattern formation and adaptation". In: *Advanced Engineering Sciences* 54.

Stenz, Anthony (May 1994). "Optimal and Efficient Path Planning for Partially-Known Environments". In: *IEEE International Conference on Robotics and Automation*, IEEE.

Tahir, Anam et al. (2019). "Swarms of Unmanned Aerial Vehicles — A Survey". In: *Journal of Industrial Information Integration* 16.

Webots (n.d.). *http://www.cyberbotics.com*. Ed. by Cyberbotics Ltd. Open-source Mobile Robot Simulation Software. URL: http://www.cyberbotics.com.